Brent O. Hatch (5715)
HATCH, JAMES & DODGE, PC
10 West Broadway, Suite 400
Salt Lake City, Utah 84101
Telephone: (801) 363-6363
Facsimile: (801) 363-6666

Robert Silver (admitted pro hac vice)
Edward Normand (admitted pro hac vice)
BOIES, SCHILLER & FLEXNER LLP
333 Main Street
Armonk, New York 10504
Telephone: (914) 749-8200
Facsimile: (914) 749-8300

Devan V. Padmanabhan (admitted pro hac vice)
DORSEY & WHITNEY LLP
50 South Sixth Street, Suite 1500
Minneapolis, Minnesota 55402
Telephone: (612) 340-2600
Facsimile: (612) 340-2868

Stephen N. Zack (admitted pro hac vice)
BOIES, SCHILLER & FLEXNER LLP
Bank of America Tower – Suite 2800
100 Southeast Second Street
Miami, Florida 33131
Telephone: (305) 539-8400
Facsimile: (305) 539-1307

Stuart Singer (admitted pro hac vice)
BOIES, SCHILLER & FLEXNER LLP
401 East Las Olas Blvd.
Suite 1200
Fort Lauderdale, FL 33301
Telephone: (954) 356-0011
Facsimile: (954) 356-0022

Attorneys for Plaintiff, The SCO Group, Inc.

**IN THE UNITED STATES DISTRICT COURT
FOR THE DISTRICT OF UTAH**

THE SCO GROUP, INC. Plaintiff/Counterclaim-Defendant, v. INTERNATIONAL BUSINESS MACHINES CORPORATION, Defendant/Counterclaim-Plaintiff.	**DECLARATION OF MARC ROCHKIND IN SUPPORT OF SCO'S MOTION FOR RECONSIDERATION OF THE ORDER DENYING SCO'S MOTION FOR RELIEF FOR IBM'S SPOLIATION OF EVIDENCE** Case No. 2:03CV0294DAK Honorable Dale A. Kimball Magistrate Judge Brooke C. Wells

1. I was retained by counsel to SCO in May 2005, to analyze the technical evidence in this case, to help prepare the preliminary October and December 2005 Disclosure of Material (the "December Submission") and to serve as a consultant and expert witness. I have since been asked to review certain issues related to the denial by the Magistrate Court of SCO's Motion for Relief for IBM's Spoliation of Evidence, and my conclusions regarding those issues are set forth herein. My qualifications are set forth in my May 19, 2006 report submitted in this case.

SCO's Use of CMVC to Develop Proof

2. I spent 200 to 300 hours setting up and reviewing CMVC and the AIX source code contained in it.

3. I used CMVC extensively in developing SCO's December Submission and my expert report. For example, AIX code from the CMVC system appears in Item 1, covering JFS, and Items 194-202 (Tabs 210-218) of the December Submission. These Items and their associated Tabs are clearly marked on every page as containing AIX code, along with the version numbers as placed in the files by CMVC. Tabs 210-218 alone contain 441,020 lines of source code (colored red) from System V that appears in AIX, which SCO alleges was misused.

4. I also extensively used the RCS system, which contains Dynix/ptx code.

5. Both CMVC and RCS have been extremely helpful in developing the substantial proof that IBM misused SCO's intellectual property, as presented in the December Submission and my expert report.

6. However, as set forth below, neither CMVC nor RCS is a substitute for the information that would have been contained in programmers' sandboxes.

Use of Sandboxes by Linux Programmers

7. I understand IBM represented that sandboxes were not used for Linux development. This is inconsistent with my understanding of programming. While the term "sandboxes" may not be used for Linux development, it would be incredibly difficult, if not impossible, to write and revise computer code without using a separate workspace, such as a sandbox, in which to implement and test those revisions. It is essential that all details of a proposed operating feature or patch be worked out in an environment separate from the online or official copy of the source code. This separate environment, the private programmer workspace, is often called a sandbox.

8. Thus, I believe that IBM programmers for Dynix/ptx and Linux, as well as AIX, would have had to use sandboxes, or other similar workspaces, to draft, revise and implement computer code for those systems.

9. My opinion that IBM programmers of Dynix/ptx and Linux would have used sandboxes, or similar systems, when writing code is supported by an interview with an IBM programmer, William Lee Irwin III, posted on the following internet website: http://kerneltrap.org/comment/reply/80, attached as Exhibit A hereto. The programmer said that he had been employed by Sequent and then IBM since 2000, and that he worked on Dynix/ptx, then AIX, and then Linux. When asked whether he had "any tips for the aspiring kernel hacker" (e.g., Linux programmer) he responded:

> Everyone says persistence is key, but it's not enough. One thing I've noticed is that because the kernel is responsible for maintaining the

> integrity of both data and the running system image, the cost of a failure (i.e. bugs, and as with any programming, they are numerous) is that data is lost and systems go down. A big fear to overcome is that of disrupting the proper operation of a system or losing data. When a kernel crashes it destroys data and the machine goes down, and you can't be afraid to see this happen if you're going to get anywhere. Programming is error-prone, and one must be prepared to commit errors. A stumbling block for me early on was that I was too careful and obsessed on repeatedly reviewing code to be sure the system wouldn't crash when I tried to run with it.
>
> This is ineffective. **A more effective approach appears to be creating a sandbox where the data is disposable and the system nonessential and running the code and figuring out what went wrong when the system crashes.** And it's not easy. Without much hardware assistance (requiring too much money to be practical) it's generally not possible to recover much of the system state after the event, so working around this by dumping state at the appropriate times or running within a simulator (fortunately, bochs is free as in beer) is required. Some infrastructures exist, e.g. kgdb and kdb. I'm already espousing perhaps heretical notions, but I don't care. And another thing is that reading code is harder than writing it (and debugging is harder than both but moving on) so a from-scratch rewrite of something will be easier than finding the small changes needed to fix real problems. For regular kernel hacking, rewrites aren't going to get anywhere, those who wrote the originals will scream bloody murder and those who have to call the stuff are terrified they'll have to deal with new bugs in unfamiliar code. But as a crutch for getting around not quite being able to read things it's fine. Maybe someone will come after me for saying so as there are bound to be frivolous rewrites of all kinds of things after any kind of public statement like this, but if people get off their butts and stop duplicating everyone else's merges of $VM + O(1) + misc garbage to write some actual new code, it's worth the flames.

(Ex. A) (emphasis added.) In short, in this interview, an IBM programmer recommended use of a sandbox to other Linux programmers as "a more effective approach" to programming for Linux. This supports my view that sandboxes or similar systems would have had to have been used by IBM and Sequent programmers in their work on Dynix/ptx and Linux.

Information in CMVC and RCS Versus Information in a Programmer's Sandbox

10. The CMVC and RCS systems are not substitutes for the information that would have been contained in programmers' sandboxes or similar workspaces.

11. First, CMVC was exclusively used for AIX code. Thus, copies of Dynix/ptx and Linux code deleted from programmers' sandboxes would clearly not be available in the CMVC.

12. Second, neither CMVC (as to AIX) nor RCS (as to Dynix/ptx) would show whether Linux programmers had retained AIX and Dynix/ptx on their systems when developing code for Linux, and if so, what parts of the AIX and Dynix/ptx operating systems they retained.

13. Third, a sandbox is the only place where the progression of code drafts can be viewed. For AIX code, CMVC shows the initial code that was checked out, and the final code that was checked back in, but not all the steps in between. The RCS system for Dynix/ptx provides even fewer details than CMVC. Yet it is these steps, these intermediate drafts, – saved only on programmers' sandboxes – that would have been so important to develop further proof of IBM's copying.

14. A change control system, such as CMVC or RCS, is like a library, and the AIX and Dynix/ptx files in those systems are like books in a library. Just as a library has records of (a) which books have been checked out, (b) when they were checked out, (c) who checked the books out, and (d) when they were checked back in, a change control system, such as CMVC or RCS might have records of (a) which files in AIX or Dynix/ptx were checked out, (b) when they were checked out, (c) who checked them out, and (d) when they were checked back in.

15. What a library does not record, and could not record, is what was done with the books during the time they were checked out, and what was done with any copies made of the books after they were checked back in. For example, the records in a library will not show:

a. whether the pages of book were copied while the book was checked out;

b. whether the reader retained pages he copied from the book he checked out, long after checking the book back in;

c. whether the reader later used those pages to write another book; or

d. what the progression of drafts of the readers' other book looked like – whether the first drafts similar to the library book he had copied, and then became progressively dissimilar.

16. Similarly, none of this information is present in CMVC in RCS. Like a library system, CMVC and RCS do not show what the programmers did with the AIX and Dynix/ptx files on their systems – in their sandboxes – after they checked it out, or even after they checked it back into the change control system. CMVC and RCS <u>do not show</u>:

a. whether the AIX and Dynix/ptx files that were checked out were copied or saved to the programmers' system (or sandbox);

b. whether the programmer retained the copied AIX and Dynix/ptx files on his or her system (or sandbox) long after checking the completed files back into the change control system;

c. whether the programmer ever referred back to those AIX and Dynix/ptx files when he was developing for another operating system, such as Linux;

d. what the progression of drafts of the programmers' code for the other operating system, such as Linux, looked like – whether the first drafts were similar to the AIX and Dynix/ptx files he had copied, and then became progressively dissimilar, perhaps as the author tried to hide his copying, until the final version actually bore little resemblance to the AIX and Dynix/ptx files files on which it had been based; or

e. whether other programmers made use of the checked-out code by obtaining it in already-checked-out form from the programmer who originally checked it out.

17. In contrast, a programmers' sandbox would show whether AIX and Dynix/ptx files were present and available to the programmer as he worked on Linux, and what specific files the programmers had retained. The code in a programmers' sandbox or similar workspace also would have shown his <u>initial</u> drafts of code and the progression of drafts after that initial draft, leading to the final version; these drafts would not be available anywhere else. From the progression of drafts it could be determined whether the programmers' initial drafts of code for Linux bore a greater resemblance to the copied AIX and Dynix/ptx files than the final draft that was ultimately disclosed to Linux.

18. Even by reviewing only the final contributions of code to Linux, I identified a significant instance of AIX code in Linux and multiple instances of Dynix/ptx code in Linux. The availability of sandboxes would have shown whether even more instances of direct copying existed, and would have enabled me to provide more specificity regarding the Dynix/ptx and AIX code underlying the programmers' disclosures to Linux.

Destruction of Sandboxes

19. In my experience, it would be highly unusual for management to get involved in "house-keeping" measures regarding programmers' systems, other than when a concern exists

over access to code. With today's technology and storage costs, the amount of space occupied by sandbox code is so small that I do not believe that management would take the time and trouble to help programmers economize on disk space. It is more likely that any instructions to remove a sandbox were issued because the information contained in the sandbox was deemed to be inappropriate for the programmers to have, not because they were wasting storage space. In fact, the entire Linux 2.4.0 source code occupies less that 115 megabytes of storage space, the same space as 10 digital pictures from my camera.

I declare under penalty of perjury that the foregoing is true and correct to the best of my knowledge.

March 16, 2007

[illegible]

Marc Rochkind

CERTIFICATE OF SERVICE

Plaintiff/Counterclaim-Defendant, The SCO Group, Inc., hereby certifies that a true and correct copy of the foregoing was served on Defendant/Counterclaim-Plaintiff, International Business Machines Corporation, on this 20th day of March 2007, via CM/ECF to the following:

David Marriott, Esq. (dmarrriott@cravath.com)
Cravath, Swaine & Moore LLP
Worldwide Plaza
825 Eighth Avenue
New York, New York 10019

Todd Shaughnessy, Esq. (tshaugnessy@swlaw.com)
Snell & Wilmer LLP
1200 Gateway Tower West
15 West South Temple
Salt Lake City, Utah 84101-1004

/s/ Edward Normand